COMP9319 2018s2 Assignment 2: BWT

This assignment contains two tasks: BWT encode and backward search.

Requirements

BWT encode

Your first task in this assignment is to create a special BWT encoder that will encode a file with delimiter-separated text values (similar to a CSV file) into two files: a BWT encoded file; and an auxiliary positional information file. Your BWT encoded file may carry some positional information (i.e., first record, second record, and so on; whilst the size of the BWT encoded file is the same as the original file, and the frequencies of each character in these two files are the same). The BWT encoded file must be the same size as the original text file, and the auxiliary must not be larger than the original text file. (Note that different students may have different, working schemes to capture this positional information).

Your C/C++ program, called **bwtencode**, accepts commandline input arguments:

1. the delimiter - it can be any visible ASCII alphabet (with ASCII value from 32 to 126) or a newline. If it is a newline, it will be specified as `'\n'`
2. the path to an index or temporary folder (see below for details);
3. the path to a file containing text values separated by the specified delimiter; and
4. the path to the encoded file to be generated.

Note that you may decide a unique file extension for your auxiliary positional information file, e.g., by appending `.aux` to the encoded filename, as far as you can determine its name from a given path to the encoded file. This file is used to store extra positional information, and shall not be used to store text such as the original text file or any encoded form of the orignal text file (e.g., LZW of the original text file).

Backward search

Your second task is to implement a simple BWT backward search, which can efficiently search a BWT encoded file produced by **bwtencode**.

Your C/C++ program, called **bwtsearch**, accepts commandline input arguments:

1. the delimiter;
2. the path to a BWT encoded file;
3. the path to an index folder;
4. either `-m` for the number of matching substrings (count duplicates), `-n` for the number of unique matching records, or `-a` for listing the identifiers of all the matching records (no duplicates and in ascending order), `-i` for displaying the content of the records with ids provided in the search term; and
5. a quoted query string (i.e., the search term).

The search term can be up to 512 characters. Here record identifiers are the positions of the records in the original text file. For example, the first record has id 1; the second record has id 2; and the third one has 3 and so on. To make the assignment easier, we assume that the search is case sensitive.

If `-a` is specified, using the given query string, **bwtsearch** will perform backward search on the given BWT encoded file, and output the sorted and unique record identifiers (no duplicates) of all the records that contain the input query string to the standard output, with one line (ending with a '\n') for each match.

If `-m` is specified, given a query string, **bwtsearch** will output the total number of matching substrings (count duplicates) to the standard output. The output is the total number, with an ending newline character. Similarly, **bwtsearch** will output the total number of unique matching records (do not count duplicates) if `-n` is specified.

Finally, when `-i` is specified, **bwtsearch** will perform backward search on the given BWT encoded file and search for the records with their identifiers beginning with `i` and ending with `j`, as specified in the search term as `"i j"`. It will output the exact content of each matching record (excluding the delimiter) to the standard output, plus an ending newline character. All these matching records will be output according to their record identifiers in ascending order. You may assume that `j`will never be smaller than `i` during testing, and all ids specified in the range of `"i j"` are valid. E.g., we will not test with `"2 10"` when there are only 5 records in the file.

The text file

Text files may include any visible ASCII alphabets (i.e., any character with ASCII value from 32 to 126), tab (ASCII 9) and newline. For example, a text file may look like

```
first$second$third$forth$
```

with `$` being the delimiter. Similarly, a newline can be used as the delimiter, then the file will look like:

```
first
second
third
forth
```

For simplicity, we assume that there is a delimiter at the end of the last record. We further assume that a file will have at least one record.

The index folder

For encoding, your solution is allowed to write out multiple external index files or temporary files to assist your encoding, providing that they are in total no larger than **ten times** of the size of the given text file. Please remove these temporary files before your program exits. Similarly, for search, your solution is allowed to write out external index files that are in total no larger than the size of the original text file. These index files may help to speed up the subsequent backward search and your program does not need to remove them when exits.

All index files or temporary files must be written inside the specified index folder (and please do not create subfolders inside the specified folder). You may safely assume that the specified folder exists.

If the total size of the files inside the index folder is larger than the corresponding size limit specified above, you will receive zero points for the tests that generating/using these files. You may assume that the index files will not be deleted during all the search tests for a given BWT file, and all the test BWT files are uniquely named. You may further assume that the index folder name that associated with a given BWT file will not be changed throughout the auto test. Therefore, to save the search time, you only need to generate the index files when they do not exist yet.

Example

Suppose the original file (let's call it dummy.txt) is:

```
Computers in industry|Data compression|Integration|Big data indexing|
```

Some examples:

```
%wagner> bwtencode '|' ~MyAccount/tmpFolder ~/a2/dummy.txt ~MyAccount/XYZ/dummy.bwt
%wagner> bwtsearch '|' ~MyAccount/XYZ/dummy.bwt dummyIndex -m "in"
4
%wagner>
%wagner> bwtsearch '|' ~MyAccount/XYZ/dummy.bwt dummyIndex -n "in"
2
%wagner>
%wagner> bwtsearch '|' ~MyAccount/XYZ/dummy.bwt dummyIndex -a "in"
1
4
%wagner>
%wagner> bwtsearch '|' ~MyAccount/XYZ/dummy.bwt dummyIndex -m "in "
1
%wagner>
%wagner> bwtsearch '|' ~MyAccount/XYZ/dummy.bwt dummyIndex -n "in "
1
%wagner>
%wagner> bwtsearch '|' ~MyAccount/XYZ/dummy.bwt dummyIndex -a "In"
3
%wagner> bwtsearch '|' ~MyAccount/XYZ/dummy.bwt dummyIndex -i "2 2"
Data compression
%wagner>
```

```
%wagner> bwtsearch '|' ~MyAccount/XYZ/dummy.bwt dummyIndex -i "1 3"
Computers in industry
Data compression
Integration
%wagner>
```

For encoding, you will need to investigate how to (slightly) extend the original BWT encoding to represent record positions (i.e., the record id). A small hint is given in the following example. Suppose that the original file is:

```
first$second$third$forth$
```

If you perform a normal BWT encoding on this file, the resultant BWT encoded file will be:

```
hdtderns$$tthfocfiio$rsr$
```

Alternatively, you may extend BWT encoding and have the following encoded file instead (you will need to figure out how this transformation is obtained):

```
htddenrs$$tthfocfiio$rsr$
```

Of course, some additional information about `$`'s need to be stored in the auxiliary file.

Compilation and the index folder

We will use the `make` command below to compile your solution. Please provide a makefile and ensure that the code you submit can be compiled on a CSE Linux machine, e.g., wagner. Solutions that have compilation errors will receive zero points for the entire assignment.

```
    make
```

After the `make` command above (without any arguments), it should compile and generate two executables, namely `bwtencode` and `bwtsearch`.

Your solution should **not** write out any external files other than those index files in the specified index folder. Any solution that writes out external files outside the index folder will receive zero points for the entire assignment.

Performance

Your solution will be marked based on space and runtime performance. Your soluton will not be tested against any files that are larger than 50MB.

Runtime memory is assumed to be always less than 15MB for search, and 250MB for encoding. Runtime memory consumption will be measured by `valgrind massif`with the option `--pages-as-heap=yes`, i.e., all the memory used by your program will be measured.
For encoding, any solution that runs for more than **300 seconds** on a machine with similar specification as *wagner* will be killed.
For search, any solution that runs for more than **60 seconds** on a machine with similar specification as *wagner* for the first query on a given BWT file will be killed, and will receive zero points for the queries for that BWT file. After that any solution that runs for more than **10 seconds** for any one of the subsequent queries on that BWT file will be killed, and will receive zero points for that query test. We will use the `time` command and count both the user and system time as runtime measurement.
Any solution that violates the memory requirement or timing requirement for search will receive zero points for that query test. If it violates the memory requirement or timing requirement for encoding, it will receive zero points for that test as well as those query tests depending on the encoded file generated from this test.

Documentation

You will be marked on your descriptions in README.txt of how your solution works and your index file structures. Your source code will be also inspected and marked based on readability and ease of understanding.

Assumptions/clarifications/hints

1. None of the testcases will result in outputting more than 5,000 records / matches.
2. The input filename is a path to the given text file. Please open the file as read-only in case you do not have the write permission.

3. Marks will be deducted for output of any extra text, other than the required, correct answers (in the right order). This extra information includes (but not limited to) debugging messages, line numbers and so on.
4. You can assume that the input query string will not be an empty string (i.e., "").
5. When counting the number of substring matches (i.e., with -m option), to make it easier for backward search matching, all combinations of matches should be counted. E.g., There are 2 matches of "aa" on text "aaa"; 2 matches of "ana" on "banana".
6. You are allowed to generate external index files (in the specified index folder) to enhance the performance of your solution. However, if you believe that your solution is fast enough without using index files, you do not have to generate any files. Even in such case, your solution should still accept a path to index folder as one of the input argument as specified (and should not generate an error message such as "unknown argument".
7. A record will not be unreasonably long, e.g., you will not see a record that is 5,000+ chars long.
8. Empty records may exist in the original files (before BWT). However, these records will never be matched during searching because the empty string will not be used as a search term when testing your program.
9. You may assume that the number of delimiters in a given file is significantly fewer than the total number of characters in that file (i.e., If the file has N characters, the number of delimiters are fewer than N/4).
10. At least half of the tests will be based on files that are smaller than 10MB. So to start, you should attempt this assignment without worrying about large files.
11. For performance consideration, you may assume that each encoded large file (i.e., larger than 10MB), there will be at least 9 subsequent tests following the test that index files are generated (so your index files can be reused).

Marking

This assignment is worth 100 points. Below is an indicative marking scheme:

Component	Points
Auto marking	95
Documentation	5

Bonus

Bonus marks (up to 10 points) will be awarded for the solution that achieves 100 points for the whole assignment (i.e., both encoding and backward search); and runs the fastest for encoding (i.e., the shortest total time to finish **all** encoding tests). Similarly, bonus marks (up to 10 points) will also be awarded for the solution that achieves 100 points for the whole assignment (i.e., both encoding and backward search); and runs the fastest for search (i.e., the shortest total time to finish **all**search tests). The same submission may be the fastest for both encoding and search, and both bonuses will be awarded to the same solution (i.e., up to 20 points).Note: regardless of the bonus marks you receive in this assignment, the maximum final mark for the subject is capped at 100.

Submission

Deadline: ~~Friday 5th~~Tuesday 9th October 23:59. Late submissions will have marks deducted from the maximum achievable mark at the rate of roughly 1% of the total mark per hour that they are late (i.e., 24% per day), and no submissions will be accepted after 3 days late. Use the give command below to submit the assignment:

```
    give cs9319 a2 makefile *.h *.c *.cpp README.txt
```

Please use "classrun" to check your submission to make sure that you have submitted all the necessary files.

Plagiarism

The work you submit must be your own work. Submission of work partially or completely derived from any other person or jointly written with any other person is not permitted. The penalties for such an offence may include negative marks, automatic failure of the course and possibly other academic discipline.

Assignment submissions will be examined both automatically and manually for such submissions.

Relevant scholarship authorities will be informed if students holding scholarships are involved in an incident of plagiarism or other misconduct.

Do not provide or show your assignment work to any other person - apart from the teaching staff of this subject. If you knowingly provide or show your assignment work to another person for any reason, and work derived from it is submitted you may be penalized, even if the work was submitted without your knowledge or consent. This may apply even if your work is submitted by a third party unknown to you.